LEVEL II

12

REPORT NO. 8105
NOVEMBER, 1981

Gary Perlman

12 19

TWO PAPERS IN COGNITIVE ENGINEERING:

THE DESIGN OF AN INTERFACE TO A PROGRAMMING SYSTEM

AND

MENUNIX: A MENU-BASED INTERFACE TO UNIX (USER MANUAL)

408 267

CENTER FOR HUMAN INFORMATION PROCE[SSING]
LA JOLLA, CALIFORNIA 92093
UNIVERSITY OF CALIFORNIA, SAN DIEGO

The research reported here was conducted under Contract N00014-79-C-0323, NR 157-437 with the Personnel and Training Research Programs of the Office of Naval Research, and was sponsored by the Office of Naval Research and the Air Force Office of Scientific Research. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agencies. Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 14 8105	2. GOVT ACCESSION NO. AD-A108929	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) "Two Papers in Cognitive Engineering: The Design of an Interface to a Programming System" and "Menunix: A Menu-Based Interface To Unix (User Manual)"		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gary Perlman		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0323
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Human Information Processing University of California, San Diego La Jolla, California 92093		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 157-437
11. CONTROLLING OFFICE NAME AND ADDRESS Personnel and Training Research Programs Office of Naval Research (Code 458) Arlington, Virginia 22217		12. REPORT DATE November, 1981
		13. NUMBER OF PAGES 25
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Cognitive engineering
Command-line interpreters
Computer programming systems
Documentation retrieval
Human factors
Human-machine interface design
Software Psychology

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

This report consists of two papers on MENUNIX, an experimental interface to the programs and files on the UNIX operating system. In the first paper, I discuss how the decisions about the design of MENUNIX were made: based on my intuitions and user comments, but also on psychological theory and data whenever available. MENUNIX presents both the programs and files of UNIX in ~~in~~ two menus from which users can make selections with single keypresses. The FILE menu presents the UNIX file hierarchy that allows users to organize files into directories by subject (e.g. writing and programming). The PROGRAM menu presents UNIX programs in a hierarchy organized into workbenches according to the tasks for which they are used (e.g. writing and programming) much as files can be organized in directories. Special facilities are provided for: finding out about useful commands; using variables to set options, to save commands, and to avoid typing long strings; and for editing strings (including recent commands). The second paper is a tutorial user manual for MENUNIX, in which the features of the program are more fully explained.

TWO PAPERS IN COGNITIVE ENGINEERING:

THE DESIGN OF AN INTERFACE TO A PROGRAMMING SYSTEM

AND

MENUNIX: A MENU-BASED INTERFACE TO UNIX (USER MANUAL)

Gary Perlman
Cognitive Science Laboratory
Department of Psychology
University of California, San Diego

ABSTRACT

This report consists of two papers on MENUNIX, an experimental interface to the programs and files on the UNIX operating system. In the first paper, I discuss how the decisions about the design of MENUNIX were made: based on my intuitions and user comments, but also on psychological theory and data whenever available. MENUNIX presents both the programs and files of UNIX in in two menus from which users can make selections with single keypresses. The FILE menu presents the UNIX file hierarchy that allows users to organize files into directories by subject (e.g., writing and programming). The PROGRAM menu presents UNIX programs in a hierarchy organized into workbenches according to the tasks for which they are used (e.g., writing and programming) much as files can be organized in directories. Special facilities are provided for: finding out about useful commands; using variables to set options, to save commands, and to avoid typing long strings; and for editing strings (including recent commands). The second paper is a tutorial user manual for MENUNIX, in which the features of the program are more fully explained.

Tuesday, November 17

THE DESIGN OF AN INTERFACE TO A PROGRAMMING SYSTEM

Gary Perlman

Cognitive Science Laboratory
Department of Psychology
University of California, San Diego

ABSTRACT

In this paper I discuss the design decisions made in programming MENUNIX, an experimental interface to the files and hundreds of programs of the UNIX operating system. Both programs and files are presented on the terminal screen in fixed location menus from which users can make selections with single character selectors displayed beside menu entries. MENUNIX organizes UNIX programs into a hierarchy in which related programs are grouped together into task-oriented workbenches, much like the way UNIX allows files to be grouped into directories. I first give a brief introduction to MENUNIX and then discuss how MENUNIX tries to be friendly to users by increasing the accessibility of programs for novices, increasing the speed of command construction for experts, and decreasing the probability and impact of errors. Psychological theory and data are referred to in support of design decisions.

CONTENTS

FIGURES

THE DESIGN OF AN INTERFACE TO A PROGRAMMING SYSTEM

Gary Perlman

On our version of the UNIX[1] operating system, there are about three hundred programs available, each with many options. An average user requires several years to be at ease with even a small subset of these commands. It is difficult to find programs to accomplish a task, and users are reluctant to search page-by-page through the hundreds of pages of the UNIX programmer's manual, even though it is alphabetically sorted. Some programs exist to help people find programs, but they are not as useful as the designers hoped for. A more friendly user-interface to UNIX programs is needed.

A *friendly* user-interface is one that facilitates program use by increasing the accessibility of computing power and increasing the efficiency of program use by speeding command expression while reducing the probability and impact of errors. In other words, a user-interface should tell its users what commands are available, and how to use them safely and efficiently. In this paper, I discuss some of the psychological issues involved in the design of MENUNIX (Perlman, 1981a), a menu-driven interface to the programs of the UNIX operating system.

MENUNIX is an experimental interface that tries to help people find and execute UNIX programs via the UNIX command line interpreter (shell) *sh* (Bourne, 1978). In MENUNIX often needed information is presented on users' CRT terminal screens. The goal of MENUNIX is to provide novice users with information about what commands are available and how they are used, while providing experts with an environment for efficiently executing commands. In short, MENUNIX is an attempt to provide a friendly user-interface to UNIX programs for users of all levels of expertise.

In the rest of this paper, I introduce MENUNIX and how it is used, and then discuss how it was designed to be friendly, referring to psychological theory and data where available. Three major sections include discussions of:

(1) How MENUNIX increases the accessibility of programs and their documentation (important for novices);

(2) How MENUNIX speeds the use of these facilities (important for experts);

(3) How MENUNIX reduces the probability and impact of errors.

I especially thank Mark Wallen for all his help in programming MENUNIX, and Don Norman for his suggestions on interface design, and comments on this paper. Steve Draper, Jim Hollan, and Phil Mercurio also added helpful suggestions. I also thank those at UC Berkeley for providing the *curses* (Arnold, 1980) software for the menu displays. MENUNIX was developed while I was supported by a postgraduate scholarship from the Natural Science and Engineering Research Council of Canada. This research was conducted under Contract N00014-79-C-0323, NR 157-437 with the Personnel and Training Research Programs of the Office of Naval Research, and was sponsored by the Office of Naval Research and the Air Force Office of Scientific Research. Author's address: Gary Perlman, Department of Psychology, C009, University of California, San Diego, La Jolla, California, 92093, USA.

[1]UNIX is a trademark of Bell Laboratories. The version we use is called "Berkeley UNIX 4.0" (Joy, 1981) and is an enhancement of Version 7 UNIX, developed at Bell Labs, (Richie & Thompson, 1974, 1978; Kernighan & Mashey, 1981).

MENUNIX: A Short Introduction

MENUNIX is a menu-based system. It always displays two menus from which users can make selections, each at a fixed location on the terminal screen. The FILE menu displays the files in the current working directory on the right side of the screen. The PROGRAM menu displays the programs available for specific tasks on the left. Both the FILE and PROGRAM menus are hierarchically structured, the former follows the UNIX file structure. An example of a MENUNIX display is shown in Figure 1.

The FILE Menu

UNIX supplies users with a file system in which files are accessed via the directories in which they reside. Directories are a special kind of file, and there can be sub-directories within directories, yielding a hierarchical structuring of files. MENUNIX presents the file system in the FILE menu. The name of the current working directory is displayed at the top of the FILE menu. Directories typically have more entries than can be displayed at once, so MENUNIX divides a directory into "pages" of file entries, and provides commands to move forward and backward through these. Each FILE menu entry includes the file's name, its size, its type (plain file, directory, etc.), and a description of how its access is protected, coded in standard UNIX format. An example of a FILE menu is shown in Figure 2.

Users may directly select files or directories by typing the entry number desired. The effect of selecting a FILE menu entry depends on the entry's type. Selecting a directory results in the FILE menu changing to the selected directory and updating its display. Selecting a plain file results in the user's preferred editor being called on the file. Selecting an executable plain file results in that program being run after program arguments are requested. In other words, MENUNIX tries to do the "sensible" operation on selected entries. This scheme is adopted from the Berkeley Computer Science Department's visual command shell, *vsh*.

The PROGRAM Menu

Though UNIX implementors suggest organizing *files* by making content-oriented directories, one for programming, one for writing, etc., UNIX *programs* are not clustered by function, but by the installation where they were written. Programs are kept in special directories such as **/bin/** (for *binary*), **/usr/bin/** (for *user*), **/usr/ucb/** (for *University of California, Berkeley*), and in **/usr/local/**, where locally developed programs may be kept. (UNIX directories are delimited by slashes.) The only programs kept in a meaningfully named directory are games, kept in **/usr/games/**.

MENUNIX organizes programs into a hierarchy much like the file system can be used to organize files. The idea of hierarchies of programs in menus was motivated by the UCSD Pascal Operating System (Bowles, 1980). Entries in the PROGRAM menu are either programs, or collections of programs, called *workbenches*, analogous to the distinction in the file structure between plain files and directories (that are collections of files). The concept of workbenches was motivated by Bell Laboratories work on the Programmer's Workbench (Dolotta, Haight, & Mashey, 1978), and the Writer's Workbench (Macdonald, Frase, & Keenan, 1980). The highest level PROGRAM menu workbench is called [UNIX] (workbench names are displayed in brackets). [UNIX] contains workbenches that, when selected, put users into task specific working environments such as [Writing Aids], [Programming], [Calculations], or [Information]. For example, [Writing Aids] contains all programs for composition: editing, spelling correction, style analysis, formatting, etc. Users can structure their directory systems so that for every workbench of regular use, there is a directory that corresponds to it, further adding to the concept of a working environment. This is shown in Figure 1 in which the [Writing Aids] workbench is displayed beside my directory for writing, **/csl/perlman/papers**.

The name of the current workbench is displayed at the top of the PROGRAM menu. Each line of the PROGRAM menu includes a short phrase describing the entry, and for program entries, the name of the UNIX command that will be called if it is selected. When a workbench is selected,

FIGURE 1: Working Environment for Writing

```
[Writing aids]                              /csl/perlman/papers/                   1/2
a     Analyze style (style)                 1     CausalInferenc/          144 dRWXr-xr-x
c     Count words and lines (wc)            2     ComputPgms/              112 dRWXr-xr-x
C     Count words used (tokens)             3     Discrimination/           48 dRWX------
d     Decode/Encode (crypt)                 4     ExptControl/             528 dRWXr-xr-x
e     Edit a file ($editor)                 5     IntfaceDesign/           176 dRWXr-xr-x
f     [Format text file]                    6     MenUnix/                 240 dRWXr-xr-x
h     Heading structure (headings)          7     NatArtLang/              160 dRWXr-xr-x
l     Look for word in dictionary (look)    8     ONR/                     112 dRWXr-xr-x
p     Permuted indexer (ptx)                9     Personal/                112 dRWXr-xr-x
r <-  Reference finder (pub)
s     Spelling error finder (spell)
S     Spelling corrector (correct)
t     Typo finder (typo)
w     Wordy sentence finder (diction)       You have new mail.
                                            Friday, November 6.   3:41:39
$1 diction MenUnix/design | more
$2 style MenUnix/design | more
$3 headings hh mh lh ph MenUnix/design | more
$4 correct MenUnix/design

COMMAND: pub '{probe}'
{probe}: (subject | title) = (design
Adding text: quit with ESC, select files with '_'
```

Note: The PROGRAM menu (left top) displays the [Writing Aids] workbenc.. and the FILE menu (right top) displays my directory for writing: /csl/perlman/papers. The HISTORY list (middle) displays recent commands stored in numerical variables ($1-$4). The FEEDBACK window (right middle) displays the time and informs users if mail has arrived. At the moment of this figure, a probe for a reference finding program is being entered into the line-editor (bottom).

FIGURE 2: A Page of a FILE Menu

```
/csl/perlman/                        1/2
1 bin/             320   dRWXr-xr-x
2 expt/            112   dRWXrwxr-x
3 menu/            624   dRWXr-xr-x
4 papers/          336   dRWXr-xr-x
5 personal/        208   dRWX------
6 .cshrc           952   -RW-r--r--
7 .lisprc         1960   -RW-r--r--
8 .login           553   -RW-r--r--
9 .logout           84   -RW-r--r--
```

Note: The top of the FILE menu displays the current working directory, and its page number/number of pages. Each entry includes a numerical selector, a filename, the size of the file, and its UNIX access protection code.

the PROGRAM menu display changes to that of the new workbench. When programs are selected, they are executed immediately unless arguments are needed, in which case MENUNIX asks for them with descriptive prompts, and reads the response from a visual one-line editor that allows using variables and edit commands. A sequence of PROGRAM menu selections is shown in Figure 3.

The PROGRAM menu is easily extended because it reads its workbenches from a set of files that code each entry using a simple format. Each line of a workbench contains the information about the phrase to be presented on the screen. For example, the following line tells MENUNIX to display the phrase "Edit a file" next to the selector e.

[selector:e] [display:Edit a file]

For program entries, there are codes for the program and arguments to be used, and for how the arguments should be requested. The following line instructs MENUNIX to use the editor stored in the user-set variable, **editor** (variables are accessed with a dollar sign), and to request "files to be edited" from the user as arguments.

[program:$editor] [argument:{file to be edited}]

For workbench entries, there is a code for where the information about that workbench is stored. Continuing the example, the [Writing Aids] workbench is stored in a file called "writing" that is included in the [UNIX] workbench with the line:

[selector:w] [display:Writing Aids] [menu:writing]

Because of this extendibility, new programs and workbenches can be introduced in a simple manner.

The CONTROL Menu

The commands used to control MENUNIX are held in an invisible (but retrievable) display called the CONTROL menu. This menu contains commands for changing workbenches and directories, setting and examining variables, executing commands, and obtaining documentation. This menu is not continuously displayed because of space limitations, because most users learn its contents easily, and because it can be retrieved when needed.

FIGURE 3: Workbenches Selected by Typing pcv

```
(a)    [UNIX]
       c          [Calculations]
       d          [Display Text]
       f          [File System]
       g          [Games]
       i          [Information]
       m          [Mail]
       n          [Network]
       p   <-     [Programming]
       r          [Reminder Service]
       s          [Searching, Pattern Matching]
       t          [Terminal Handler]
       w          [Writing Aids]

(b)    [Programming]
       a          APL (apl)
       c   <-     [C Programming]
       f          [FORTRAN]
       l          [LISP]
       m          MicroSOL (m.sol)
       p          [Pascal]
       t          [Tools]

(c)    [C Programming]
       c          Compiler (cc)
       d          Debugger (adb)
       e          Error handler (error)
       m          Make program (make)
       p          Pretty Printer (cb)
       v   <-     Verify program (lint)
       x          Cross-referencer (ctags)
              ...
       COMMAND: lint {options} {files}
       {options}:
```

Note: The command sequence pcv (begun in the [UNIX] workbench (a)) goes through the [Programming] (b) and [C Programming] workbenches (c) in which selections cause arrows to point to the selected option just before the display changes. In (c), the v in pcv selects the "Verifier" in the [C Programming] workbench and the MENUNIX line-editor requests options and files to be verified.

PROVIDING ACCESS

In this section I discuss how MENUNIX deals with the problem of how users can learn about options when hundreds exist.

Displaying Options in Menus Aids Memory

For MENUNIX, a *menu*, like in a restaurant, is a display of options from which users can make selections. These options might be commands, files, or program parameters. The display of options allows users to *recognize* a desired option instead of having to *recall* it. This is especially useful because there may be several possible names for an option, and users may be able to recognize one of these option names as the one that will accomplish a task, but may not be able to guess the one the programmer chose. For example, an editor might be called "editor" or "edit" or even some related term, like "compose" or "modify." The purpose of programs with such functional names is clear, but figuring out which is the real one is a matter of luck. In a non-menu-based system, recalling the name of an option is not enough; the option has to be typed in. It is not always a simple task to recall how a program's name is spelled as programs often acquire unpredictable abbreviations.

With small menus, fast displays, and efficient option selection schemes, each option can be accompanied by a short phrase describing its function, instead of a single word or a cryptic abbreviation. Our editors have names like: ed, ex, vi, and emacs. The phrase, "Edit a file" gives users a clear indication of the function of the program (edit), as well as a clue about what argument the program expects (a file). A method of choosing the most retrievable names for objects has been developed by Baggett (1980) and was used to generate some descriptive phrases for UNIX programs in the PROGRAM menu.

Hierarchical Structuring Facilitates Discovery

Given that a menu with several hundred options is not practical, what is the best way to divide options into many smaller menus? People tend to organize information into what Miller (1956) called "chunks," each of which has a capacity for about seven sub-chunks. Cognitive psychology research in the 1960s showed that the optimal organization for material to be learned is hierarchical (Mandler, 1967, 1968, 1970). MENUNIX follows these lessons by organizing programs into a hierarchy in which commands with similar functions are grouped together into *workbenches*. Each workbench contains about ten options, all in the semantic category defined by the name of the workbench. Programs and workbenches that serve multiple purposes are multiply represented in the PROGRAM menu hierarchy, thereby increasing the accessibility of programs. (Strictly speaking, this last feature makes the PROGRAM menu not a real hierarchy.)

Users are reminded of old programs and discover new ones because each workbench is a cognitively manageable size. Commands can be learned as they are needed for specific tasks. For example, new users, who are mainly interested in learning how to use the mail system and edit files, can learn exactly those commands of interest through the appropriate workbenches, and can easily ignore the others, except when they deliberately explore. The PROGRAM menu has proven to be a valuable means of discovering new commands because users can browse through content oriented workbenches.

Controlling workbench size. Workbenches are kept at a cognitively manageable size (about ten entries) by using sub-workbenches. For example, the [Programming] workbench contains over fifty programs that are distributed in sub-workbenches for programming in [C], [LISP], [Pascal], [FORTRAN], and for general program development [Tools].

Providing Documentation

Once an option has been found, a user may want to read documentation about it before using it. MENUNIX uses an operation developed by Bobrow and Winograd (1977) that allows users to take different *perspectives* on any program, including CONTROL menu commands. Ordinarily, selecting a program option results in its execution, but users can make MENUNIX adopt the perspective that selecting a program fetches information about it. In this manner, MENUNIX attempts to provide assistance with all commands at all points in the PROGRAM menu hierarchy.

INCREASING THROUGHPUT

To increase throughput, memory and perceptual aids can be used to speed the task of specifying requests, and variables can be used to avoid typing often-used long strings.

Fixed-Location Tabular Formats Speed Menu Search

To find an option in one of many menus, first the appropriate menu has to be found, and then that menu has to be searched. If menus are displayed at fixed locations on a terminal screen, they will be relatively easy to find. Some programs display their menus by appending them to the bottom of the screen, causing the screen to scroll and make it difficult to predict the location of particular menus. MENUNIX reserves fixed areas of the screen for PROGRAM and FILE menus, and for presenting feedback and for editing, and this helps users find displays faster. Text presented in tabular formats make use of Gestalt grouping principles of *proximity* and *continuation* and is easier to scan (Horn, 1981), especially if options can be arranged in some familiar way, such as alphabetical sorting (Perlman, 1981b).

Option Selection Schemes

Once the command that performs a desired function is known, it has to be selected. One scheme for selecting items in a menu is by moving the cursor (with key controls, light−pen, mouse, etc.) to the location of the desired item, and pressing a "select" key. This is fine for novices, who depend on menus to find commands, but impedes experts who know the commands they want and do not wish to find them in a display and move cursors.[2] For an average display, even experts take several seconds to find and point to any option (Perlman, 1981b). Ideally, the same selection scheme serves all levels of users so there can be a smooth transition from novice to expert use. Any scheme involving pointing will not meet this requirement, so single character selectors were chosen for MENUNIX because they can be executed quickly and are easily memorized.

[2]This is not to say that pointing devices are never useful because they are more *generally* useful than the scheme described here. Single character selectors are best suited for this particular application.

Static display selectors. A good alternative to pointing is to select options by a letter in a keyword related to its function. For example, most people easily remember to select the compiler with **c**. and the debugger with **d**. (There is no confusion about which compiler or debugger is wanted because of the previous choice of workbenches). Using mnemonic selectors is possible with *static* displays in which the options do not change appreciably over time, such as workbenches of programs. If more than one program description phrase begins with the same letter, synonyms or rewording can be used to avoid selector clashes. For example, in the [Writing Aids] workbench, there are two programs that begin with **s**, *spell* (spelling error finder), and *style* (style analysis). *Spell* can be displayed with "Correct spelling" and selected with **c**, or *style* can be displayed with "Analyze style" and selected with **a**. This process shows another advantage of using descriptive phrases over single names: programs can easily be renamed to optimize selector choice.

Dynamic display selectors. Using mnemonic selector letters is not possible with *dynamic* displays in which the options can be expected to change continually, such as a file directory whose entries change daily. This is because several options may demand the same selector, such as files that begin with the same letter, and users would have to attend to the screen while conflicts are resolved. It may seem advantageous to use alphabetized letters instead, but pairing sorted letters and options produces incompatibility. For example, to select a file named "paper" with the letter **d** is difficult because the name "paper" activates the letter **p** that competes with the correct selector. In MENUNIX, the solution is to use more neutral selectors: sorted numerals. Using sorted numerals as selectors has some advantage because an option's selector can be determined from two sources: the character displayed next to an option, and the option's ordinal position in the display. The latter source is most advantageous for options toward the beginning of lists, so the FILE menu displays the most used options, directories, before any others in the display. (Perlman, 1981b).

Hierarchical Structuring

Because programs are hierarchically organized in the PROGRAM menu, and each PROGRAM menu entry is selected with a single mnemonic letter, mnemonic strings coding the path through general to specific workbenches ending with program selection can be memorized. For example, the C program verifier can be selected with the string "pcv" because the **p** selects the [Programming] workbench, the **c** selects the [C Programming] workbench, and the **v** calls the "Verifier." (This name is easier to reconstruct than the program's real name, *lint*, so called because it "picks up pieces of fluff" in programs.) This sequence is shown in Figure 3. These path-names tend to be short because of the hierarchical structure of the PROGRAM menu, usually two or three characters, so they are executed quickly by experts. (Intermediate menus are not displayed when a command sequence is typed quickly.) The hierarchical structuring also speeds novice use because the time it takes people to search through a hierarchy is logarithmic in the number of entries, compared to linear time for large sorted displays (Perlman, 1981b).

Variables and Macros

Variables and macros allow users to customize MENUNIX so commonly used strings can be replaced by short ones. MENUNIX provides facilities for defining variables before entering MENUNIX, and for setting them while MENUNIX is running.

Default valued variables. MENUNIX provides users with a set of predefined variables, such as their home directory, their mail spool file, and the current working directory. These provided variables can be used in other variables or in constructing commands.

Commonly accessed directories. To change directories, MENUNIX users can change their working directories to filenames entered in MENUNIX's line-editor. Since UNIX pathnames can be long, variables can be used instead, and this greatly speeds the execution of this common command. For example, this paper is being written in a directory called **/csl/perlman/papers/menu**, and I have a variable called **pm** (short for *papers/menu*), set to that long name. To work on this paper, I simply type: **/pm** and the FILE menu displays my files for this paper.

Commonly used commands. MENUNIX allows users to execute commands stored in variables, and so allows users to have faster access to commonly used or personal commands. In the Berkeley UNIX command line interpreter (shell), *csh*, this command substitution is called *aliasing*. Wherever a command may be used (at the beginning of a line or after command separators), MENUNIX checks for variables and substitutes their values. A commonly used alias in MENUNIX is the **editor** variable that tells which editor is to be used when plain files are selected from the FILE menu.

Recently used commands. MENUNIX keeps a record of recently executed commands, and allows them to be accessed to be edited or re-executed with numerical variable names. In the Berkeley command shell, a special mechanism called the *history list* is implemented for this purpose. In MENUNIX, a few of the most recent of these are displayed on the middle lines of the MENUNIX display as a reminder to users (see Figure 1). For users to re-execute a stored command, they can type the command number instead of a command, and the aliasing mechanism substitutes the stored command for the numeral.

Entering and Editing Information

MENUNIX allows its users to modify variables and recent commands in a one-line editor similar to *open* mode in the Berkeley *vi* and *ex* editors. This facility allows users to combine commands, or modify ones with minor mistakes, without taking the time to reconstruct them.

While in the line-editor, users are asked for information with prompts specific to each program represented in the PROGRAM menu hierarchy. This speeds the process of entering information because it reminds users of what is required. For most programs, the most likely argument is the name of a file, so MENUNIX provides special facilities for speeding the supplying of filenames. Files can be specified using regular expressions because MENUNIX filters its commands through the standard UNIX shell *sh* (Bourne, 1978) that provides this facility. MENUNIX allows users to yank the names of files into its line-editor by going into a special "file selection" mode in which filenames are inserted by pressing their selector keys. For example, if I wanted to supply the names ".login," ".menuvar," and ".remind" to a program (see Figure 2), I could go into file-selection mode and type "678," and the filenames would be yanked into the string I was editing. Because of these filename supplying facilities, filenames are seldom typed, and tend to be longer and more meaningful, and therefore easier to recognize.

Focusing Attention with Workbenches and Directories

MENUNIX helps users focus their attention on tasks, thereby speeding their completion, and allows short diversions to other tasks. By displaying the task specific programs in a workbench, the user is reminded of the task at hand, and by displaying the files in a directory, the subject of the task is present. Because recently executed commands are displayed, users are reminded of what they have been working on. This is shown in Figure 1.

Diversions to secondary tasks. MENUNIX allows short diversions from a working environment, during which time it remembers its state and automatically returns to it after program execution. For example, if users are notified in a feedback window that new mail has arrived and want to read it, MENUNIX allows them to hit a "diversion" key that puts them into the [UNIX] workbench that allows them to use an absolute command key sequence (beginning from the [UNIX] workbench; for example, "mm" goes into the [Mail] workbench and into the UCSD "Message" system) to access a desired program. After execution of this program, MENUNIX automatically returns to its pre-diversion state, reminding users of what they were working on.

Reducing Wasted Commands by Providing Feedback

From an analysis of typical system use with a non-menu interface, about 15% of all commands are used to answer questions about the time and date, whether any mail has been received, what commands have been executed recently, the directory where a user is working, and what options a user has (such as the files available). With a visual display, MENUNIX reserves fixed locations of the screen to answer these questions so users do not have to execute commands to ask them (see Figure 1).

REDUCING ERRORS

Simplicity and Consistency

Simplicity of commands refers to how many concepts must be learned to understand how something works and is desirable for error-free use. In MENUNIX, users execute commands simply by pressing a key next to the desired command, thereby reducing the probability of mistyping commands. MENUNIX reduces the concepts users have to deal with by representing *sh variables*, and the *csh history list* and *aliases* with a single more general class of variables. Experienced users of UNIX who have become accustomed to separate concepts of aliasing, history list, and variables, find this confusing, but there is no evidence for this confusion among users raised on MENUNIX (though there is a lack of such a population).

Consistency between commands is usually necessary for error-free use because if commands use similar concepts inconsistently, users applying analogies to learn or use commands will make mistakes (Perlman, in press). The syntaxes of commands do not have to be consistent when executed via MENUNIX, because it asks users for program arguments with meaningful prompts, making argument order relatively unimportant.

Providing Feedback and Prompts

MENUNIX provides several types of feedback to users besides the PROGRAM and FILE menus that tell users about available commands. A real-time clock ticking tells users MENUNIX is running properly (and reminds them of their inactivity). MENUNIX prompts users for any information that is needed to run a command. For example, to copy files, the UNIX *cp* commands takes two sets of arguments: the first is a list of files to be copied, and the second is their destination. When the "Copy" command in the [File System] workbench is selected, users are prompted for "files to be copied" and "destination," and so have a clear idea of what is required of them at any time, reducing the possibility of making the error of supplying the right information at the wrong time.

The Problem of Modes

A *mode error* happens when users think a program is in one mode when it is in another, and try to use a command in the mistaken mode (Norman, 1981). For example, users may forget they are in an editor command-mode, think they are in an append-mode, try to add text, and become confused as the editor mistakes their text for commands, possibly with bad results. Mode errors are minimized in MENUNIX by avoiding modes where possible, or by providing users dynamic feedback of the current mode. If feedback is not dynamic, users may commit errors by believing old feedback. For example, if files removed by number are still displayed, removing the third file in a list a second time may really remove the fourth.

Unique use of symbols. The use of modes has been avoided to some degree in MENUNIX, and when a mode is entered, some feedback is provided, though that solution is not ideal, as experts tend to ignore predictable feedback.[3] Modes were avoided in the choice of selectors for the FILE menu (exclusively numbers), PROGRAM menu (exclusively letters), and CONTROL menu (exclusively symbols) by using the convention that no selectors for any menus would overlap.

Despite this precaution, some errors occur when users press a key for one menu that has a similar effect in another menu. For example, one can "pop" out of workbenches or directories, and users have been observed typing the command for popping workbenches to pop directories, and *vice versa*. Norman (1981) calls this a *description error:* formulating the correct intention, but forming an insufficiently specific description (Norman & Bobrow, 1979), leading to ambiguity in the selection of the correct response.

Absolute command sequences. Modes can also be avoided by always preceding commands by the key that puts the [UNIX] workbench in the PROGRAM menu. From the [UNIX] workbench, the sequence of keys to access any command is unique, and is called an ***absolute command sequence***, analogous to using absolute path names for specifying UNIX files. If commands are not so anchored, users may commit mode errors. For example, if users think they are in the [File System] workbench, but are in the [UNIX] workbench, and want to "protect access" to their files by typing **p**, they will be surprised to find themselves in the [Programming] workbench.

---

[3]Users ignoring feedback can cause what Norman (1981) calls *capture errors*, errors that occur when users act out of force of habit and do a common action when an unusual one is required.

Menu Selection Errors

The scheme MENUNIX uses for selecting menu options is based on data on how long it takes and how hard it is to pair selectors with options (Perlman, 1981b). While mnemonic (first letter) selectors are most desirable because their use results in shorter selection times and fewest errors, it is not possible to use them for menus that have options that change often, such as the FILE menu. The Berkeley visual command shell, *vsh*, uses the sorted letters a-t to select from a list of sorted files. However, this scheme results in what Norman (1981) calls *activation errors* because when a file like "paper" is selected with l, the name "paper" activates the letter p and the competing activations slow correct responses, and cause some selection errors (Perlman, 1981b).

CONCLUSION

An empirical analysis of the successes and failures of MENUNIX has yet to be made, partly because of the difficulty in obtaining data for such a complex system under controlled conditions. Though many of the design decisions were made on the basis of psychological theory and data, the true test of the usefulness of a program is only found empirically. Based on watching and getting feedback from a few users, I can offer some observations. Because of its complexity, MENUNIX is imposing at first, but usually a ten minute personal introduction starts a new user on the right track. This seems to be true for most complex menu-oriented systems and it may be that such programs need special programs written just to help new users. One unexpected drawback of MENUNIX is that it is easy to get used to. Having a lot of information displayed on a screen is very comforting to users, and being prompted for most inputs makes going back to a less interactive system difficult.

MENUNIX is a continuing project with which I am studying user-interface design issues. In these studies, I have tried to address Norman's (in press) criticisms of the consistency of command names and syntax, the functionality of the command names, and the "friendliness" to users of UNIX. Though derived from experience with MENUNIX, I think the results apply equally well to other systems as the design issues for all programs overlap considerably. MENUNIX helps UNIX novices by introducing them to UNIX programs as they need them, and by helping them run the programs by providing descriptive prompts for information as it is needed. It is also useful to expert UNIX users who prefer an interface that requires few keypresses and provides many reminders. That it can accommodate both novices and experts suggests that MENUNIX approximates a friendly user-interface, and can influence the design of others like it.

REFERENCES

Arnold, K. C. R. C. Screen Updating and Cursor Movement Optimization: A Library Package. Berkeley, California: Department of Electrical Enginerring and Computer Science, University of California, Berkeley. 1980.

Baggett, P. Comprehension of verbal/pictorial instructions. Boulder, Colorado: Department of Psychology, University of Colorado (Boulder). September 9, 1980. (Office of Naval Research Quarterly Report.)

Bobrow, D. G., & Winograd, T. An overview of KRL, A knowledge representation language. *Cognitive Science*, 1977, 1, 3-46.

Bowles, K. L. *Beginner's manual for the UCSD Pascal system.* San Diego, California: Institute for Information Sciences. 1980.

Dolotta, T. A., Haight, R. C., & Mashey, J. R. The programmer's workbench. *Bell System Technical Journal*, 1978, 57, 2177-2200.

Horn, R. E. Structured writing and text design. In D. H. Jonassen (Ed.), *The technology of text.* New York: Educational Technology Press, in press.

Joy, W. N. *UNIX programmer's manual: Seventh edition (Virtual VAX-11 version).* Berkeley, California: Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. 1980.

Kernighan, B. W., & Mashey, J. R. The UNIX programming environment. *Computer*, 1981, 14, 12-22.

Macdonald, A. H., Frase, L. T., & Keenan, S. A. Writer's workbench: Computer programs for text editing and assessment. Murray Hill, New Jersey: Bell Laboratories. May 9, 1980. (Bell Tech. Memo TM-80-3771-2 49580.)

Mandler, G. Organization and memory. In K. W. Spence, & J. T. Spence (Eds.), *The psychology of learning and motivation: Advances in research and theory.* New York: Academic Press, 1967.

Mandler, G. Association and organization: Facts, fancies and theories. In T. R. Dixon, & D. L. Horton (Eds.), *Verbal behavior and general behavior theory.* Englewood Cliffs, New Jersey: Prentice Hall, 1968.

Mandler, G. Words, lists, and categories: An experimental view of organized memory. In J. L. Cowan (Ed.), *Studies in thought and language.* Tucson, AZ: University of Arizona Press, 1970.

Miller, G. A. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 1956, 63, 81-97.

Norman, D. A., & Bobrow, D. Descriptions: An intermediate stage of memory retrieval. *Cognitive Psychology*, 1977. 11, 107-123.

Norman, D. A. Categorization of action slips. *Psychological Review*, 1981, 88, 1-15.

Norman, D. A. The truth about UNIX: The user interface is horrid. *Datamation*, in press.

Perlman, G. Making mathematical notation more meaningful. *The Mathematics Teacher*, in press.

Perlman, G MENUNIX: A menu-driven interface for UNIX. La Jolla, California: University of California, San Diego. August 24, 1981. (Unpublished Cognitive Science Laboratory user manual.)

Perlman, G. Pointing by numbers: Putting the finger on keys to response compatibility. La Jolla, California: University of California, San Diego. August, 1981. (Cognitive Science Laboratory paper in preparation.)

Richie, D. M., & Thompson, K. The UNIX time-sharing system. *Communications of the ACM*, 1974, 17, 365-375.

Richie, D. M., & Thompson, K. The UNIX time-sharing system. *Bell System Technical Journal*, 1978, 57, 1905-1929.

MENUNIX: A MENU-DRIVEN INTERFACE TO UNIX PROGRAMS (USER MANUAL)

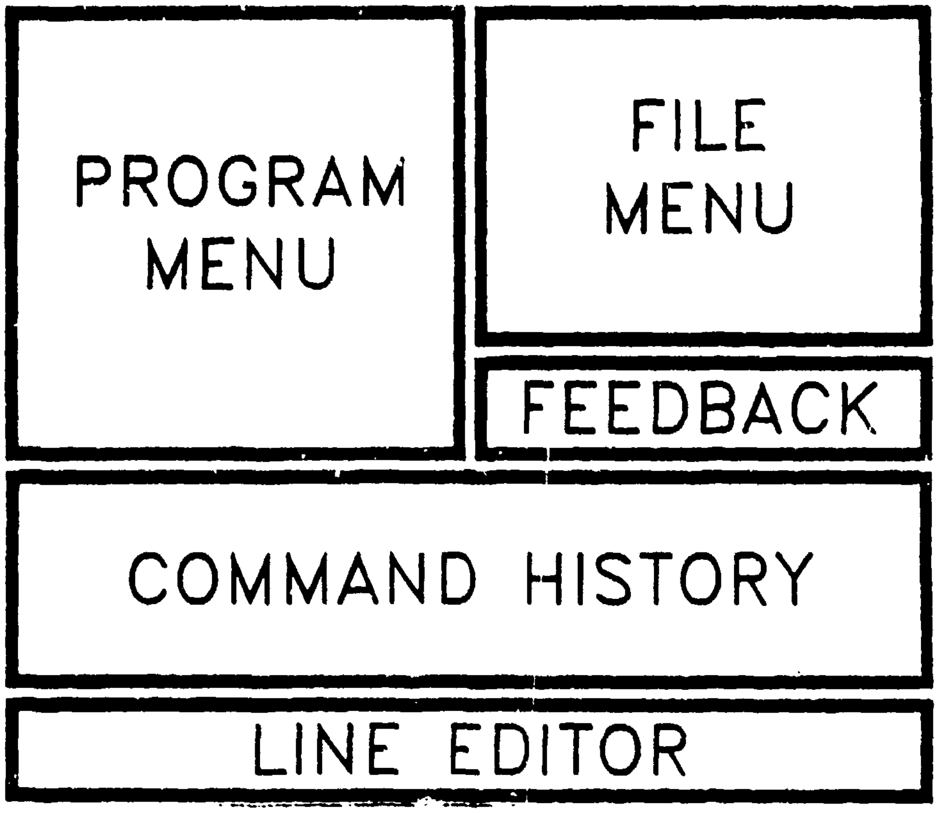
Gary Perlman

Cognitive Science Laboratory
Department of Psychology
University of California, San Diego

ABSTRACT

This is a manual for MENUNIX, an interface to programs and files on the UNIX operating system. Programs and files are displayed in menus on users' terminal screens, and are selected with single keypresses of characters displayed next to menu entries. The FILE menu presents the UNIX file hierarchy in menu format, and commands for moving through the hierarchy in both absolute and relative terms are provided. The PROGRAM menu organizes UNIX commands into a hierarchy in which related programs are grouped together into task oriented workbenches, analogous to files being grouped into directories. Special commands are provided for setting, examining, and using variables via a one-line editor that also allows modifying commands. The PROGRAM menu hierarchy, like the FILE menu hierarchy, can be customized to the needs of individual users, allowing workbenches for unusual tasks such as linear programming in addition to ones for common tasks like writing.

CONTENTS




```
[Writing aids]                              /csl/perlman/papers/                1/2
a     Analyze style (style)                 1    CausalInferenc/       144 drwxr-xr-x
c     Count words and lines (wc)            2    ComputPgmg/           112 drwxr-xr-x
C     Count words used (tokens)             3    Discrimination/        48 drwx------
d     Decode/Encode (crypt)                 4    ExptControl/          528 drwxr-xr-x
e     Edit a file ($editor)                 5    IntfaceDesign/        176 drwxr-xr-x
f     [Format text file]                    6    MenUnix/              240 drwxr-xr-x
h     Heading structure (headings)          7    NatArtLang/           160 drwxr-xr-x
l     Look for word in dictionary (look)    8    ONR/                  112 drwxr-xr-x
p     Permuted indexer (ptx)                9    Personal/             112 drwxr-xr-x
r <-  Reference finder (pub)
s     Spelling error finder (spell)
S     Spelling corrector (correct)
t     Typo finder (typo)
w     Wordy sentence finder (diction)       You have new mail.
                                            Friday, November 6.  3:41:59
$1 diction MenUnix/design | more
$2 style MenUnix/design | more
$3 headings hh mh lh ph MenUnix/design | more
$4 correct MenUnix/design

COMMAND: pub '{probe}'
{probe}: (subject | title) = (design
Adding text: quit with ESC, select files with '_'
```

MENUNIX: A MENU-DRIVEN INTERFACE TO UNIX PROGRAMS (USER MANUAL)

Gary Perlman

The UNIX[1] menu system, MENUNIX, provides a structured environment for executing UNIX commands. MENUNIX presents menus from which you can select programs and files displayed on a terminal screen. A diagram of the MENUNIX display is shown in the figure at the beginning of this manual. The PROGRAM MENU on the left side of the screen lets you select programs, and the FILE MENU on the right side lets you select files. Each of the menu display entries can be selected with a single key, displayed on the left of each entry. The effects of selecting PROGRAM and FILE MENU entries will be described in following sections.

The FILE MENU

On the right side of the MENUNIX display is the FILE MENU, that displays the UNIX file system. At its top is the full pathname of the current working directory, followed by the page number and number of pages of that directory (3/5 means you are on page three of a total of five). Directories typically have more entries than can be displayed on a screen, so each directory is divided into pages, each containing a manageable sized part. At most nine files are displayed at one time, and these are selected with the number keys 1−9. You can leaf through the pages of a directory by using the plus and minus keys. Typing a plus goes to the next page (if there is one), and typing a minus goes to the previous one. Trying to go to a page that is not there causes MENUNIX to "wrap around" to the first or last page of a directory.

To select a file, you type the number on the left of its name. If you type a number without any file beside it (this can happen on the last page of a directory), MENUNIX will beep at you. In general, MENUNIX beeps when you type a character that is not acceptable. The effect of selecting a file depends on the type of file selected, that can be determined from the information displayed on the right of its name. To the right of a file is displayed its size (in characters), and its access protection modes.

The access protection modes contain ten characters interpreted as follows. The first character indicates the type of file.

-	plain file
b	block-type special file
c	character-type special file
d	directory
B	multiplexor-type block special file
C	multiplexor-type character special file

The next nine characters are interpreted as three sets of three bits. The first set refers to access permissions of the owner of the file, the next three to permissions to others in the same user-group, and the last three to permissions to all others. Within each set, the three characters respectively indicate permission to read, write, or execute the file as a program. These permissions are coded as:

r	the file is readable
w	the file is writable
x	the file is executable
-	the indicated permission is not granted

The owner of the file is coded by capital read, write, and execute flags. The owner of a file sees the first three bits in capital letters, the members of the owner's group see the second three bits in

---

[1] UNIX is a trademark of Bell Laboratories.

capital letters, and all others see the last set of three bits in capitals. In any case, the operations one can perform on a file will be in capital letters.

Selecting a directory causes the working directory to change to that directory. Directories are identified by their access modes, by being followed by a slash, and on some terminals by being displayed more prominently. Selecting an executable file (a program) causes that program to be run after arguments are requested. Selecting a plain file calls the editor named by the variable **editor** on that file (or the *ex* editor if it is not set). (See the later section on how to set variables.)

The number key 0 is not used to select a file but is reserved for going "upwards" in the file system. Pressing the number key to the left of a directory name goes into the selected directory, and the 0 key lets you pop out of a directory.

In summary, the commands for the FILE MENU are:

0	Changes the working directory to the parent directory.
1-9	Select the FILE MENU entry beside the number typed. This edits files, executes programs, and changes directories.
+	Displays the next page of the working directory.
-	Displays the previous page of the working directory.

The PROGRAM MENU

On the top left of the MENUNIX display is the PROGRAM MENU. The PROGRAM MENU organizes programs into a hierarchy much as the file system can be used to organize files. Entries in the PROGRAM MENU are either programs, or collections of programs, called *workbenches*. This is analogous to entries in the FILE MENU being files, or collections of files, called directories. At the top of the PROGRAM MENU is the name of the workbench in use. The initial workbench is [UNIX] (workbenches are always displayed enclosed in [square] brackets). [UNIX] contains the most general classification of UNIX commands, so general that it does not contain any programs, but only names of task specific workbenches that contain all and only those commands that are of interest to people working on a specific task. For example, there is a programming workbench (that contains more specific workbenches for C programming, FORTRAN, LISP, Pascal, and general program development tools. Other high level workbenches are:

Calculations	Calculators and statistics
Display	Displaying and formatting files
File System	Programs to change files
Games	The fate of the weak
Information	System status and documentation
Mail	Sending and receiving mail
Network	Intermachine communication
Reminder Service	Get daily reminders of events
Searching, Pattern Matching	Regular expressions, Grammars, etc.
Writing Aids	Spelling, style analysis, etc.

Each line of the PROGRAM MENU consists of a short phrase describing an entry (program or workbench) that is selected by typing the mnemonic letter on the left of its name. Selecting a workbench causes the PROGRAM MENU to change to and display the one selected. To get back to the [UNIX] workbench, you can type a period, or repeatedly type commas that pop up one workbench at a time. Program entries are displayed followed by the parenthesized name of the UNIX program they call. How a program executes depends on whether it needs information, such as the name of a file. If no information is needed, the screen clears and the program runs. After the program is done, MENUNIX will ask you to type a RETURN if a program is expected to produce some output you might want to read before the screen is cleared and the MENUNIX display reappears.

When a program requires that information be specified, MENUNIX asks for that information in a one-line editor located at the bottom of the screen. For example, in the [File system] workbench is an entry for copying files. Selecting that program causes a prompt like:

{files to be copied}:

to appear in the line editor. When a program needs information, a prompt for what is needed is placed in {curly} braces, for which you should type in the information (followed by a RETURN to tell it you are finished). Immediately above the line where you type in the requested information is the command that MENUNIX is generating. For the file copying program, this command looks like:

cp {files to be copied} {destination}

This means that MENUNIX needs two bits of information from you. After you supply the {files to be copied} information, MENUNIX will ask you for the {destination}. The one-line editor is fully described in a later section.

The [CONTROL] Menu: Controlling MENUNIX

There is one workbench, [CONTROL], that is important because its entries can be accessed at any time, no matter what workbench is being displayed. It contains commands for controlling the PROGRAM and FILE MENUs, and also some commands you are likely to need at any time, such as your preferred editor, set with the **editor** variable. [CONTROL] contains commands for changing directories, changing directory pages, changing workbenches, stopping MENUNIX, and so on. Since these commands are used so often, they become memorized and their display becomes unnecessary. The [CONTROL] workbench can be seen at any time by "flipping" to it with an ampersand, &. Typing & changes the display in the PROGRAM MENU without changing the commands available, as though the two workbenches were one on top of the other. The [CONTROL] workbench options are described below.

Summary of MENUNIX Internal Commands

NAME	*CHAR*	*FUNCTION*
ampersand	&	flips to/from [CONTROL]
period	.	goes to [UNIX] (first) workbench
colon	:	goes to [UNIX] workbench and returns
comma	,	goes to the parent workbench
zero	0	goes to the parent directory
slash	/	changes directory to that specified in the editor
plus	+	displays the next directory page
minus	-	displays the previous directory page
question	?	changes to/from the documentation perspective
exclamation	!	runs the command supplied in the editor
dollar	$	sets a variable to a value
pound	#	prints the value of a string or variables
CTRL-r	^R	redisplays the screen

Stopping MENUNIX

BREAK or **DEL** causes MENUNIX to ask if you really want to quit. You can type a second **BREAK** or **DEL** to quit, or type a RETURN to return to MENUNIX.

Redisplaying the Screen

If the screen gets messed up for some reason, you can redisplay it by typing **CTRL-r**.

Changing Workbenches

A workbench can be entered by typing the letter beside its name. You can leave that workbench by "popping" up to its parent workbench (the workbench you were in before entering it) with a comma. You can pop up to the [UNIX] workbench by typing a period.

A special operation is supplied to allow you to take a short diversion from what you are working on. For example, MENUNIX may inform you that you have new mail, and you may want to read it and immediately return to your task. You can pop up to the [UNIX] workbench and have MENUNIX remember where you were so that it will return after you execute a command by popping up to the [UNIX] workbench with a colon. It also can be used to stop your history list of commands from getting cluttered with unimportant commands because commands executed in this manner are not saved. For example, you may be programming and notice you have mail, and you could type ":mm" which would read your mail and return you to your old workbench automatically. You can make an early return to the saved workbench by typing a second colon.

Setting Variables

You can set a variable to a value to be used at a later time. This is done by typing the character on the left of the "set variable" command in [CONTROL], the dollar sign, $. The variable setting command will ask you for a {name} (that must be made entirely of letters), and then a {value} (that can have any content, even other variables). Variables can be edited using the one-line editor described later.

Preset variables. There are some variables that are set when you go into MENUNIX. They can be used in your responses to MENUNIX prompts by preceding their names by $. The predefined variables are:

user	your login name
home	your login directory
dir	the current directory
mail	your mail spool file
menu	the menu source directory

User defined variables. When it starts, MENUNIX checks in your home (login) directory for a file called **.menuvar** from which it will read any variables you may want defined. Variables are set by lines in this file of the form:

NAME = VALUE

where NAME is any uninterrupted string of upper or lower case letters, and VALUE is any text including predefined variables. For example, you may want to have fast access to directories you often use, and define the following variables.

p = $home/papers
pm = $p/menu
pp = $p/personal

Examining variables. You can find the value of a variable by printing a string that includes them. To print the value of a string, (this is called interpolation), the pound sign, #, command is offered in [CONTROL]. Typing # results in MENUNIX prompting you for a string at the bottom of the screen. If you type in

$menu

MENUNIX will print the name of the directory holding the definition of the PROGRAM MENU hierarchy on the line above. This directory is discussed in a later section. If you type RETURN when # asks you for a string, MENUNIX will print all the variable names and values.

Using variables as commands. There are nine variables, 1−9, that hold the most recently executed commands. 1 is the most recently executed command, 2 is the next most recently executed command, and so on. A few of the most recently executed commands are displayed in the middle of the MENUNIX display. You can execute a recently used command by executing the variable set to it. Like any variable, you can edit these in the one-line editor described later.

Executing Commands

By typing the "Execute command" selector, !, you can enter any command you would type in the shell, *sh*, outside MENUNIX. All commands constructed in MENUNIX are executed via *sh* so commands can include pattern matching expressions (such as *) and redirection and piping of inputs and outputs. Executing commands with ! is useful for repeating a command stored as a variable on the history list (variables 1−9). As an example, suppose that the variable 3 held the command:

sort foo.bar

and that you want to save the result of that command in a file in your home directory. You could type a !, and MENUNIX would ask you for a command to execute, and you would type in:

$3 > $home/sorted

which would sort foo.bar and put the result in the file called "sorted" in your home (login) directory.

Instead of typing $3 > $home/sorted, you can omit the first $ because MENUNIX checks the first field of all commands and replaces it with the value of the variable by that name (if it exists).

Executing a command with ! does not update the history variables 1-9, but the variable 0 is always used to construct commands and so can be accessed to rerun any command.

Changing Directory

Besides 0 (zero), which pops you up one directory, and selecting directories from the FILE MENU, you can change directories with the "Change directory" command, the slash, /. Typing a slash makes MENUNIX prompt you for the desired new directory. You can type the full pathname of the directory you want to enter. This can include variables, as usual. You can also type RETURN, that has the same effect as typing $home; both return you to your home directory.

A common use of variables is to alias directory names, and because of this, the directory changing command allows you to change directory directly to a variable name. So if $foo is /usr/lib/tmac, you can type / to change directories, and foo to tell MENUNIX where to go; the $ is not needed.

Getting Documentation

By typing the Execution/Documentation character, ?, you change perspectives on the PROGRAM MENU. Ordinarily, selecting a program causes its execution, but by typing a ?, you switch into a mode in which the next PROGRAM MENU program you select will cause MENUNIX to look for documentation on that program, even if it is a MENUNIX [CONTROL] command. Once you get documentation on a program, MENUNIX automatically puts you back in the execution perspective. The ? is really a toggle for changing perspectives, so if you go into the documentation perspective and you want out, another ? changes you back.

Entering Information: MENUNIX's One-Line Editor

Many commands require you to supply information, such as the names of file arguments, or option setting flags. To do this, MENUNIX has you enter information in a one-line editor, called Line-edit, located at the bottom of the screen, Line-edit allows you to include and delete characters from anywhere inside a line you are editing, as well as insert variables in responses. When MENUNIX puts you in Line-edit, it is generally to provide some information for a command it is going to be running. MENUNIX automatically starts you in "appending text" mode; everything you type is entered into a buffer. When in "append" mode, you can enter text and follow with a RETURN, and MENUNIX will receive what you have typed. This will be a common use of Line-edit, however there are times when you will want to change something you have typed, or perhaps a variable or recent command, and you will want to get into the middle of a line and make changes. For this, Line-edit has "cursor mode" in which you can move the cursor to any point in the line and make changes.

Moving the cursor. In cursor mode, you can move to the right or left with the keys labeled with arrows (if your terminal is so equipped). An **l** (letter 'el') moves you one character forward (as does a space, +, or CTRL-l), and an **h** moves you one back (as does backspace and -). Capital letters tend to apply to a whole line rather than just a character. An L moves you to the far right of the line, an H to the far left. You can move forward or backward a word at a time with **w** or **b** respectively.

Adding new text. To append text after the cursor, type **a**, and to append text after the end of the line, type A. To insert text before the cursor, type **i**, and to insert text before the beginning of the line, type I. Minor mistakes can be corrected by backspacing. Once in an adding text mode, you can return to cursor mode by typing the key labeled ESC (for escape). Alternatively, you can type RETURN and MENUNIX will immediately read what you have typed.

File selection mode. In an adding text mode, you can go into a file selection mode in which the names of files are added to your edit line as you type the selector numbers beside their names. File selection mode is entered by typing the file selection character, the underscore, _. In this mode, every time you type the number beside a file name, that file name is added to your edit line. To stop this mode, you can repress the underscore, which will return you to the editor in an adding text mode, or press RETURN to send your edit line to MENUNIX.

Mistakes. In cursor mode, typing an **x** removes the character under the cursor. A zero, **0**, deletes the contents of the editor from the cursor to the end of the line. A capital **X** clears the whole line and automatically puts you into append mode. Any mistake you have just made can be undone by pressing **u** which gives you back your edit line as it was before the last change. If you have really messed things up, you can type **U** which gives you the line you began editing, which is unfortunately often nothing.

Stopping line-edit. A RETURN will always send what you have typed to MENUNIX, regardless of mode. In cursor mode, a **q** can also be used to quit editing. If you do not want MENUNIX to look at what you have typed, say to abort a command, you can type **Q**. Also, if you are really desperate, you can type **BREAK**, and MENUNIX will ask you if you want to quit MENUNIX completely.

Summary of Line-Edit Commands

a	append text after the cursor
A	append text after the end of the line
b	back up one word
h	move the cursor back one character
H	move the cursor to the beginning of the line
CTRL-h	go back one character and delete if adding text
l	move the cursor forward one character
L	move the cursor to the end of the line
q	leave Line-Edit and pass back contents
Q	leave Line-Edit and stop command
u	undo the last change
U	undo all changes
w	forward one word
x	delete the character below the cursor
X	delete the contents of the editor and insert
0	delete from the cursor to the end of the line
+	move the cursor forward one character
-	move the cursor back one character
_	enter or leave file selection mode
	ignore the special meaning of the next character
ESC	stop adding text

The Menu Definition Language

Just as you have control over the files in your file system, you can change the structure of the PROGRAM MENU hierarchy. A predefined variable in MENUNIX is menu that holds the name of the directory holding files that define the PROGRAM MENU hierarchy, that contains two special files: UNIX which defines the [UNIX] workbench and all subsidiary workbenches; and CONTROL defines the [CONTROL] workbench of commands available at all parts of MENUNIX.

In $menu is a file called UNIX that has lines that define

(1) The name of each entry in the [UNIX] workbench,

(2) The one character selector for that entry,

(3) The type of the entry (whether the entry is that of a workbench or a program). For program entries, arguments may be supplied.

Each part of a workbench entry is defined by a bracketed field of the form:

[NAME:VALUE]

where NAME specifies the name of the field, and VALUE specifies its value. For example, the [UNIX] workbench has the definition:

[display:UNIX][selector:.][menu:UNIX]

This definition says that "UNIX" should be displayed on the right of the selector character '.' and that its selection will cause the display of a menu whose definition is in the directory $menu in a file called "UNIX." An example of a program entry is that for the copy command:

[disp:Copy files][sel:c][prog:cp][args:{files} {destination}]

which says that "Copy files" should be displayed on the right of the selector character "c" and that its selection will cause the execution of the UNIX *cp* command.

c Copy files (cp)

Since there is an argument field, MENUNIX knows to append it to the call to *cp*. Anything in the argument field is interpolated and copied unless there is a part of the field enclosed in {curly} braces. MENUNIX uses the convention that anything in curly braces is to be used as a prompt to get a response from the user. For each of the braced parts of the argument VALUE field, MENUNIX presents that part to the user and replaces it with the interpolated response typed in.

As a summary, each line of a workbench file defines a workbench entry that is either for another workbench (defined in an other file), or a program that may have arguments that the user may have to supply. Each entry is divided into [NAME:VALUE] fields. The names of these fields (that may be abbreviated to just one character) are listed below, along with a description of their uses.

display Defines what is displayed.

selector Defines the character to be used to select the entry.

menu Defines that the entry is that of a workbench menu. the VALUE field holds the name of the file in $menu that contains the definition of the menu.

program Defines the name of the UNIX program to be executed when the entry is selected. If the entry is for a workbench, this field is ignored.

arguments Supplies information to be appended to a UNIX command defined by the program field. This information can be regular text, including variables, which is interpolated and appended, or it can be enclosed in {curly} braces, which is replaced by the interpolated response obtained from the user after presenting the braced pattern.

waitoff Tells MENUNIX to clear the screen and redisplay without user permission after a UNIX program has been executed. Without this field, MENUNIX asks permission with a prompt. The waitoff field has no value.

Defining the [CONTROL] Workbench

The [UNIX] workbench is defined by a special file in the $menu directory, $menu/UNIX. Another special file, used to define the [CONTROL] workbench, is $menu/CONTROL. The definition for $menu/CONTROL is just like any other workbench, but the commands in [CONTROL] are available at all parts of the PROGRAM MENU. This is because MENUNIX searches menus in a specific order for the selector character typed. First MENUNIX sees if the user has typed any of the numbers 1−9, used to access file entries. Then MENUNIX checks [CONTROL], and finally the current workbench. This means that the numbers 1−9 are permanently reserved, and that any characters in [CONTROL] should be carefully selected because they will not be available for any other menus.

The programs that are used in [CONTROL] should also be carefully selected because only fifteen entries are allowed. The entries should be reserved especially for the commands MENUNIX uses to control the display, called internal commands whose names are preceded by a minus. The internal commands available are listed below. After the letter is its default selector character in parentheses. In $menu/CONTROL, the selectors for these internal commands are defined, so if you don't like using a selector, you can choose your own.

MENUNIX Internal Commands

u(.)	changes the workbench to [UNIX]
a(:)	changes the workbench to [UNIX] and returns
f(&)	flips the PROGRAM MENU display to [CONTROL]
p(,)	changes the workbench to the parent menu
0(0)	changes the working directory to the parent directory
c(/)	changes directory
+(+)	displays the next directory page
-(-)	displays the previous directory page
d(?)	changes to and from the documentation perspective
s(!)	runs a command typed in the line-editor
v($)	sets a variable to a value
i(#)	prints the value of a string or prints all variable values
r(CTRL-r)	redisplays the screen

Making Your Own Menu System

Changing $menu makes it possible to customize the PROGRAM MENU hierarchy to your liking. when you fire up MENUNIX, you can add an argument to the program call that sets **menu.** This must be the complete pathname of the directory with the files defining the PROGRAM MENU hierarchy. To make your own PROGRAM MENU hierarchy, you would create a directory with the files UNIX and CONTROL, which refer to other files (containing workbenches) in the directory you supplied to the call to MENUNIX. A good way to begin is to copy all the files from the standard $menu to your preferred $menu, and then make modifications.

Navy

1 Dr. Arthur Bachrach
Environmental Stress Program Center
Naval Medical Research Institute
Bethesda, MD 20014

1 CDR Thomas Berghage
Naval Health Research Center
San Diego, CA 92152

1 Dr. Alvah Bittner
Naval Biodynamics Laboratory
New Orleans, LA 70189

1 Chief of Naval Education and Training
Liason Office
Air Force Human Resource Laboratory
Flying Training Division
Williams AFB, AZ 85224

1 CDR Mike Curran
Office of Naval Research
800 N. Quincy St.
Code 270
Arlington, VA 22217

1 Dr. Pat Federico
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. John Ford
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Richard Gibson
Bureau of Medicine and Surgery
Code 3C13
Navy Department
Washington, DC 20372

1 LT Steven D. Harris, MSC, USN
Code 6021
Naval Air Development Center
Warminster, Pennsylvania 18974

1 Dr. Lloyd Hitchcock
Human Factors Engineering
Division (6022)
Naval Air Development Center
Warminster, PA 18974

1 Dr. Jim Hollan
Code 304
Navy Personnel R & D Center
San Diego, CA 92152

1 CDR Charles W. Hutchins
Naval Air Systems Command Hq
AIR-340F
Navy Department
Washington, DC 20361

1 CDR Robert S. Kennedy
Head, Human Performance Sciences
Naval Aerospace Medical Research Lab
Box 29407
New Orleans, LA 70189

1 Dr. Norman J. Kerr
Chief of Naval Technical Training
Naval Air Station Memphis (75)
Millington, TN 38054

1 Dr. William L. Maloy
Principal Civilian Advisor for
Education and Training
Naval Training Command, Code 00A
Pensacola, FL 32508

1 Capt. Richard L. Martin, USN
Prospective Commanding Officer
USS Carl Vinson (CVN-70)
Newport News Shipbuilding and Drydock Co
Newport News, VA 23607

1 Dr. George Moeller
Head, Human Factors Dept.
Naval Submarine Medical Research Lab
Groton, CN 06340

1 Dr. William Montague
Navy Personnel R & D Center
San Diego, CA 92152

1 Commanding Officer
U.S. Naval Amphibious School
Coronado, CA 92155

1 Ted M. I. Yellen
Technical Information Office, Code 201
Navy Personnel R&D Center
San Diego, CA 92152

1 Library, Code P201L
Navy Personnel R&D Center
San Diego, CA 92152

1 Technical Director
Navy Personnel R&D Center
San Diego, CA 92152

6 Commanding Officer
Naval Research Laboratory
Code 2627
Washington, DC 20390

1 Psychologist
ONR Branch Office
Bldg 114, Section D
666 Summer Street
Boston, MA 02210

1 Psychologist
ONR Branch Office
536 S. Clark Street
Chicago, IL 60605

1 Office of Naval Research
Code 437
800 N. Quincy Street
Arlington, VA 22217

1 Office of Naval Research
Code 441
800 N. Quincy Street
Arlington, VA 22217

5 Personnel & Training Research Programs
(Code 458)
Office of Naval Research
Arlington, VA 22217

1 Psychologist
ONR Branch Office
1030 East Green St.
Pasadena, CA 91101

1 Office of the Chief of Naval Operations
Research Development & Studies Branch
(OP-115)
Washington, DC 20350

1 LT. Frank C. Petho, MSC, USN (Ph.D)
Selection and Training Research Div.
Human Performance Sciences Dept.
Naval Aerospace Medical Research Lab.
Pensacola, FL 32508

1 Roger W. Remington, Ph.D
Code L52
NAMRL
Pensacola, FL 32508

1 Dr. Bernard Rimland (03B)
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Worth Scanland, Director
Research, Development, Test & Eval.
Naval Education and Training
Code N-5
NAS, Pensacola, FL 32508

1 Dr. Sam Schiflett, SY 721
Systems Engineering Test Directorate
U.S. Naval Air Test Center
Patuxent River, MD 20670

1 Dr. Robert G. Smith
Office of Chief of Naval Operations
OP-987H
Washington, DC 20350

1 Dr. Alfred F. Smode
Training Analysis & Evaluation Group
(TAEG)
Dept. of the Navy
Orlando, FL 32813

1 Dr. Richard Sorensen
Navy Personnel R&D Center
San Diego, CA 92152

1 Roger Weissinger-Baylon
Dept. of Admin. Sciences
Naval Postgraduate School
Monterey, CA 93940

1 Dr. Robert Wherry
562 Mallard Drive
Chalfont, PA 18914

1 Dr. Robert Wisher
Code 309
Navy Personnel R&D Center
San Diego, CA 92152

1 Mr John H. Wolfe
Code P310
U. S. Navy Personnel Research and
Development Center
San Diego, CA 92152

Army

1 Technical Director
U.S. Army Research Institute for the
Behavioral and Social Sciences
5001 Eisenhower Ave.
Alexandria, VA 22333

1 Dr. Beatrice J. Farr
U.S. Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333

1 Dr. Robert Sasmor
U. S. Army Research Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

Air Force

1 U.S. Air Force Office of
Scientific Research
Life Sciences Directorate, NL
Bolling Air Force Base
Washington, DC 20332

1 Air University Library
AUL/LSE 76/443
Maxwell AFB, AL 36112

1 Dr. Earl A. Alluisi
HQ, AFHRL (AFSC)
Brooks AFB, TX 78235

1 Dr. Genevieve Haddad
Program Manager
Life Sciences Directorate
AFOSR
Bolling AFB, DC 20332

1 David R. Hunter
AFHRL-MOAM
Brooks AFB, TX 78235

Marines

1 Special Assistant for Marine
Corps Matters
Code 100M
Office of Naval Research
800 N. Quincy St.
Arlington, VA 22217

1 Dr. A.L. Slafkosky
Scientific Advisor (Code RD-1)
HQ, U.S. Marine Corps
Washington, DC 20380

CoastGuard

1 Chief, Psychological Reserch Branch
U. S. Coast Guard (G-P-1/2/TP42)
Washington, DC 20593

Other DoD

12 Defense Technical Information Center
Cameron Station, Bldg. 5
Alexandria, VA 22314
Attn: TC

1 Military Assistant for Training and
Personnel Technology
Office of the Under Secretary of Defense
for Research & Engineering
Room 3D129, The Pentagon
Washington, DC 20301

1 DARPA
1400 Wilson Blvd.
Arlington, VA 22209

Civil Govt

1 Dr. Paul G. Chapin
Linguistics Program
National Science Foundation
Washington, DC 20550

1 Dr. Susan Chipman
Learning and Development
National Institute of Education
1200 19th Street NW
Washington, DC 20208

1 William J. McLaurin
66610 Howie Court
Camp Springs, MD 20031

1 Dr. H. Wallace Sinaiko
Program Director
Manpower Research and Advisory Services
Smithsonian Institution
801 North Pitt Street
Alexandria, VA 22314

1 Dr. Joseph L. Young, Director
Memory & Cognitive Processes
National Science Foundation
Washington, DC 20550

Non Govt

1 Dr. John R. Anderson
Dept. of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Anderson, Thomas H., Ph.D
Center for the Study of Reading
174 Children's Research Center
51 Gerty Drive
Champaign, IL 61820

1 Dr. John Annett
Dept. of Psychology
University of Warwick
Coventry CV4 7AL
England

1 Dr. Michael Atwood
Science Applications Institute
40 Denver Tech. Center West
7935 E. Prentice Ave.
Englewood, CO 80110

1 Psychological Research Unit
Dept. of Defense (Army Office)
Campbell Park Offices
Canberra ACT 2600, Australia

1 Dr. Alan Baddeley
Medical Research Council
Applied Psychology Unit
15 Chaucer Rd.
Cambridge CB2 2EF
England

1 Dr. Patricia Baggett
Dept. of Psychology
University of Denver
University Park
Denver, CO 80208

1 Dr. Jonathan Baron
Dept. of Psychology
University of Pennsylvania
3813-15 Walnut St. T-3
Philadelphia, PA 19104

1 Mr Avron Barr
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Jackson Beatty
Department of Psychology
University of California
Los Angeles, CA 90024

1 Liaison Scientists
Office of Naval Research
Branch Office, London
Box 39 FPO New York 09510

1 Dr. Lyle Bourne
Department of Psychology
University of Colorado
Boulder, CO 80309

1 Dr. John S. Brown
XEROX Palo Alto Research Center
3333 Coyote Road
Palo Alto, CA 94304

1 Dr. Bruce Buchanan
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. C. Victor Bunderson
WICAT INC.
University Plaza Suite 10
1160 So. State St.
Orem, UT 84057

1 Dr. Pat Carpenter
Dept. of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. John B. Carroll
Psychometric Lab
Univ. of No. Carolina
Davie Hall 013A
Chapel Hill, NC 27514

1 Dr. William Chase
Dept. of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Dr. Micheline Chi
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

1 Dr. William Clancey
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Allan M. Collins
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

1 Dr. Lynn A. Cooper
LRDC
University of Pittsburgh
3939 O'Hara St.
Pittsburgh, PA 15213

1 Dr. Meredith P. Crawford
American Psychological Association
1200 17th Street, N.W.
Washington, DC 20036

1 Dr. Kenneth B. Cross
Anacapa Sciences, Inc.
P.O. Drawer Q
Santa Barbara, CA 93102

1 Dr. Diane Damos
Arizona State University
Tempe, AZ 85281

1 Dr. Emmanuel Donchin
Department of Psychology
University of Illinois
Champaign, IL 61820

1 LCOL J. C. Eggenberger
Directorate of Personnel Applied Research
National Defence HQ
101 Colonel By Drive
Ottawa, Canada K1A 0K2

1 ERIC Facility-Acquisitions
4833 Rugby Avenue
Bethesda, MD 20014

1 Dr. A. J. Eschenbrenner
Dept. E422, Bldg. 81
McDonnell Douglas Astronautics Co.
P.O.Box 516
St. Louis, MO 63166

1 Mr. Wallace Feurzeig
Bolt Beranek & Newman, Inc.
50 Moulton St.
Cambridge, MA 02138

1 Dr. Edwin A. Fleishman
Advanced Research Resources Organ.
Suite 900
4330 East West Highway
Washington, DC 20014

1 Dr. John R. Frederiksen
Bolt Beranek & Newman
50 Moulton Street
Cambridge, MA 02138

1 Dr. Alinda Friedman
Dept. of Psychology
University of Alberta
Edmonton, Alberta
Canada T6G 2E9

1 Dr. R. Edward Geiselman
Dept. of Psychology
University of California
Los Angeles, CA 90024

1 Dr. Robert Glaser
LRDC
University of Pittsburgh
3939 O'Hara St.
Pittsburgh, PA 15213

1 Dr. Marvin D. Glock
217 Stone Hall
Cornell University
Ithaca, NY 14853

1 Dr. Daniel Gopher
Industrial & Management Engineering
Technion-Israel Institute of Technology
Haifa
ISRAEL

1 Dr. James G. Greeno
LRDC
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

1 Dr. Harold Hawkins
Department of Psychology
University of Oregon
Eugene OR 97403

1 Dr. Barbara Hayes-Roth
The Rand Corporation
1700 Main Street
Santa Monica, CA 90406

1 Dr. Frederick Hayes-Roth
The Rand Corporation
1700 Main Street
Santa Monica, CA 90406

1 Dr. James R. Hoffman
Dept. of Psychology
University of Delaware
Newark, DE 19711

1 Dr. Earl Hunt
Dept. of Psychology
University of Washington
Seattle, WA 98015

1 Dr. Ed Hutchins
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Steven W. Keele
Dept. of Psychology
University of Oregon
Eugene, OR 97403

1 Dr. David Kieras
Dept. of Psychology
University of Arizona
Tuscon, AZ 85721

1 Dr. Walter Kintsch
Dept. of Psychology
University of Colorado
Boulder, CO 80302

1 Dr. Kenneth A. Klivington
Program Officer
Alfred P. Sloan Foundation
630 Fifth Ave.
New York, NY 10111

1 Dr. Stephen Kosslyn
Harvard University
Department of Psychology
33 Kirkland St.
Cambridge, MA 02138

1 Dr. Marcy Lansman
Dept. of Psychology
NI-25
University of Washington
Seattle, WA 98195

1 Dr. Jill Larkin
Dept. of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Dr. Alan Lesgold
Learning R & D Center
University of Pittsburgh
Pittsburgh, PA 15260

1 Dr. Robert R. Mackie
Human Factors Research, Inc.
5775 Dawson Ave.
Goleta, CA 93017

1 Dr. Mark Miller
TI Computer Science Laboratory
C/O 2825 Winterplace Circle
Plano, TX 75075

1 Dr. Allen Munro
Behavioral Technology Laboratories
1845 Elena Ave., Fourth Floor
Redondo Beach, CA 90277

1 Dr. Seymour A. Papert
Massachusetts Institute of Technology
Artificial Intelligence Lab
545 Technology Square
Cambridge, MA 02139

1 Dr. James A. Paulson
Portland State University
P.O. Box 751
Portland, OR 97207

1 Dr. James W. Pellegrino
University of California,
Santa Barbara
Dept. of Psychology
Santa Barbara, CA 93106

1 Mr. Luigi Petrullo
2431 N. Edgewood Street
Arlington, VA 22207

1 Dr. Martha Polson
Department of Psychology
Campus Box 346
University of Colorado
Boulder, CO 80309

1 Dr. Peter Polson
Dept. of Psychology
University of Colorado
Boulder, CO 80309

1 Dr. Steven E. Poltrock
Dept. of Psychology
University of Denver
Denver, CO 80208

1 Dr. Mike Posner
Department of Psychology
University of Oregon
Eugene, OR 97403

1 Dr. Diane M. Ramsey-Klee
R-K Research & System Design
3947 Ridgemont Drive
Malibu, CA 90265

1 Mr. Riley
LRDC
University of Pittsburgh
3939 O'Hara St.
Pittsburgh, PA 15213

1 Dr. Andrew M. Rose
American Institutes for Research
1055 Thomas Jefferson St. NW
Washington, DC 20007

1 Dr. Ernst Z. Rothkopf
Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974

1 Dr. Walter Schneider
Dept. of Psychology
University of Illinois
Champaign, IL 61820

1 Dr. Robert J. Seidel
Instructional Technology Group
HUMRRO
300 N. Washington St.
Alexandria, VA 22314

1 Committee on Cognitive Research
c/o Dr. Lonnie R. Sherrod
Social Science Research Council
605 Third Ave.
New York, NY 10016

1 Robert S. Siegler
Associate Professor
Carnegie-Mellon University
Dept. of Psychology
Schenley Park
Pittsburgh, PA 15213

1 Dr. Edward Smith
Bolt, Beranek & Newman, Inc.
50 Moulton St.
Cambridge, MA 02138

1 Dr. Richard Snow
School of Education
Stanford University
Stanford, CA 94305

1 Dr. Robert Sternberg
Dept. of Psychology
Yale University
Box 11A, Yale Station
New Haven, CT 06520

1 Dr. Albert Stevens
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

1 David E. Stone, Ph.D.
Hazeltine Corporation
7680 Old Springhouse Rd.
McLean, VA 22102

1 Dr. Patrick Suppes
Institute for Mathematical Studies in
the Social Sciences
Stanford University
Stanford, CA 94305

1 Dr. Kikumi Tatsuoka
Computer Based Education Research
Laboratory
252 Engineering Research Laboratory
University of Illinois
Urbana, IL 61801

1 Dr. David Thissen
Department of Psychology
University of Kansas
Lawrence, KS 66044

1 Dr. Perry Thorndyke
The Rand Corp.
1700 Main St.
Santa Monica, CA 90406

1 Dr. Douglas Towne
University of So. Calif.
Behavioral Technology Labs
1845 S. Elena Ave.
Redondo Beach, CA 90277

1 Dr. Benton J. Underwood
Dept. of Psychology
Northwestern University
Evanston, IL 60201

1 Dr. Phyllis Weaver
Graduate School of Education
Harvard University
200 Larsen Hall, Appian Way
Cambridge, MA 02138

1 Dr. David J. Weiss
N660 Elliott Hall
University of Minnesota
75 E. River Rd.
Minneapolis, MN 55455

1 Dr. Keith T. Wescourt
Information Sciences Dept.
The Rand Corporation
1700 Main St.
Santa Monica, CA 90406

1 Dr. Susan E. Whitely
Psychology Dept.
University of Kansas
Lawrence, Kansas 66044

1 Dr. Christopher Wickens
Dept. of Psychology
University of Illinois
Champaign, IL 61820

1 Dr. J. Arthur Woodward
Department of Psychology
University of California
Los Angeles, CA 90024